# 附录 各章习题参考答案

## 第1章 习题参考答案

1. 答：面向对象技术以对象为基础，使用对象抽象现实世界中的事物，以消息来驱动对象执行处理。具有以下特点：维护简单，面向对象程序设计的一个特征就是模块化；实体可以被表示为类以及同一名字空间中具有相同功能的类，可以在名字空间中添加一个类而不影响该名字空间的其他成员，为程序的维护提供了便捷性；可扩充，如果有一个具有某一种功能的类，就可以扩充这个类，创建一个具有扩充功能的类；易代码重用，功能是被封装在类中的，类是作为一个独立实体而存在的，因此可以很简单的提供类库，使代码得以重复使用。

2. 答：面向过程程序中算法决定了程序的流程以及函数间的调用关系，也就是函数之间的相互依赖关系。算法和数据结构二者相互独立，分开设计。面向对象程序设计中数据和函数封装在一起，形成对象。对象传递消息，以完成所需任务。各个对象的操作完成了，系统的整体任务也就完成了。在面向对象程序设计中，所有人都使用相同的概念和表示方法。而且，要处理的概念和表示方法都比较少。数据和过程不是人为分离的。数据和过程一起放在容易管理的小软件包中，数据从来不与算法分开。最后得到的代码不太复杂，对客户需求的变化也不太敏感。

3. 答：捕获系统需求。使用的表示法应该可以被用户和开发者理解。通过开发合适的分析模型分析系统。使用恰当的表示法表达模型，这样开发者就可以快速、方便地从中提取信息。 开发系统设计。使用恰当的表示法来开发和表达设计模型，使设计师和程序员都可以理解。系统设计师可能要操作分析模型，并在此过程中做出设计决策。实现、测试和部署系统。这也需要使用合适的表示法来表示这些活动的工件，使之可以被系统设计师、程序员和系统测试人员所理解。

4. 答：对象是对问题域中某些事情的抽象，反映了系统保持信息，并且与信息交互，或者二者兼有的能力。

5. 答：类是具有相同属性和操作的一组对象的集合，它为属于该类的全部对象提供了统一的抽象描述。事实上，类与对象的关系如同模具和铸件之间的关系。类是对象的抽象定义，或者说是对象的模板，对象是类的实例，或者说是类的具体表现形式。类封装了一组对象的公共属性。

6. 答：封装是面向对象编程的特征，用于分析模型。指的是在对象中放置数据，同时将操作应用于数据。对象其实就是一些数据以及处理这些数据的操作。数据存储于对象的特性中，数据和特性一起组成了对象的信息结构。

7. 答：信息隐藏比封装更进了一步，使得对象不可能以任何方式访问另一对象的数据，除非调用操作。这避免了每个对象都需要“知道”其他对象的所有外部细节。

本质上，对象只需要知道自身的数据和操作。然而，很多处理过程也因此必须包括知道如何请求来自其他对象的服务。服务是对象或子系统执行的代表其他对象或子系统的动作，包括对存储于其他对象中的数据的索引。此时，对象必须“知道”该向其他对象提问什么，以及如何提问。但是也不需要对象“知道”其他对象发布服务方式的所有信息。这一“知识”要求负责实现对象的程序员有其他对象实现方式的详细信息。

8. 答：继承是面向对象编程语言实现一般化和特殊化的一种机制。当两个类通过继承机制进行关联的时候，更一般化的类称为超类，而另一个相关的类，更特殊化的称为子类。

9. 答：多态的字面意思是“表现为多种形式的状态”，是指相同的消息被发送到不同类对象的可能性，每一个类对象都会以不同、但是合理的方式响应消息。这意味着原始对象不需要知道在特定场合即将接收消息的类。多态中关键的一点是，每一个对象都知道如何响应接收到的有效消息。同一条消息被不同的对象接收到时可能产生完全不同行为，这就是多态性。多态性支持“同一接口，多种方法”的面向对象原则，使高层代码只写一次而在低层可以多次复用。

实际上，在现实生活中可以看到许多多态性的例子。如学校发布一条消息：8月25日新学期开学。不同的对象接收到该条消息后会做出不同的反应：学生要准备好开学上课的必需物品，教师要备好课，教室管理人员要打扫干净教室，准备好教学设施和仪器，宿舍管理人员要整理好宿舍等。显然，对于同一条消息，不同的接收对象做出了不同的反应，这就是多态性。可以设想，如果没有多态性，那么学校就要分别给学生，教师，教室管理人员和宿舍管理人员等许多不同对象分别发开学通知，分别告知需要做的具体工作，显然这是一件非常复杂的事情。有了多态性，学校在发消息时，不必一一考虑各种类型人员的特点，而不断发送各种消息，只需要发送一条消息，各种类型人员就可以根据学校事先安排的工作机制有条不紊的工作。

10. 答：UML（Unified Modeling Language，统一建模语言 ）是一种建模语言，是用来为面向对象开发系统的产品进行说明可视化和编制文档的建模方法。在面向对象编程中，数据被封装(或绑定)到使用它们的函数中，形成一个整体称为对象，对象之间通过消息相互联系。面向对象建模与设计是使用现实世界的概念模型来思考问题的一种方法。对于理解问题、与应用领域专家交流、建模企业级应用、编写文档、设计程序和数据库来说，面向对象模型都非常有用。

11. 答：计算机系统是一个由硬件、软件组成的多级层次结构，它通常由微程序级、一般机器级、操作系统级、汇编语言级、高级语言级组成，每一级上都能创造程序设计，且得到下级的支持。

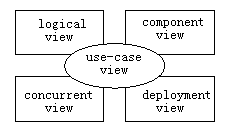
12. 答：Rational Rose、PowerDesigner、Visio是三种常用的UML建模工具。其中，Rational Rose支持全系列的UML模型图，而且很容易体现迭代、用例驱动等特性，相关性最好，缺点是图形质量差，逻辑检查与控制差，没有Name和Code的区分（PowerDesigner的特性）不太适合中国人，生成文档不好也不适合自定义，也没有设计对象的字典可以快速查找。PowerDesigner也支持全系列的UML模型图，优点是图形质量好，生成的文档容易自定义，逻辑检查与控制好，有设计对象的字典可以快速查找和快速在图形中定位，缺点是相互之间的衔接比较麻烦，对UML和RUP不熟练的人使用PowerDesigner，体现不出来迭代和用例驱动。相比较起来Visio的图形质量是最好的，但是衔接和相关性也是最差的，逻辑检查和控制也比较差。

另外，好的建模工具软件支持模型文档与代码、模型文档与数据库之间的双向转换。常用的UML工具Rational Rose通过中间插件能够实现文档与代码、数据库的双向转换，该功能是通过中间插件实现的。PowerDesigner支持模型文档与代码、模型文档与数据库之间的双向转换最好，而且不需要插件。Visio通过VBA和宏实现模型文档与代码、模型文档与数据库之间的双向转换，用起来稍微麻烦。

Rational Rose有相对最新最完整UML支持，PowerDesigner和Visio稍微滞后一点。Rose有RUP体系的支持和一系列支持RUP的软件与Rose协作，这一点是PowerDesigner和Visio望尘莫及的。

## 第2章 习题参考答案

1. 答：UML 中的视图包括用例视图（Use-case view）、 逻辑视图（Logical view ）、并发视图（Concurrency View）、构件视图（Component view ）和部署视图（Deployment View） 等五种，它们之间的关系如图所示。



2. 答：UML定义了九种不同类型的图：用例图、类图、对象图、序列图、协作图、状态图、活动图、构件图、部署图。一个简单的图画并不能完全反映出系统中需要的所有信息。描述一个系统涉及到该系统的许多方面，比如功能性方面（它包括静态结构和动态交互）、非功能性方面（可靠性、扩展性和安全性等）和组织管理方面（工作组、映射代码模块等）。完整地描述系统，通常的做法是用一组视图反映系统的各个方面，每个视图代表完整系统描述中的一个抽象，显示这个系统中的一个特定的方面。每个视图由一组图构成，图中包含了强调系统中某一方面的信息。

3. 答：UML扩展机制主要包括版类（stereotype ）、加标签值（tagged value） 和约束（constrains）。版类扩展机制是指在已有的模型元素基础上建立一种新的模型元素。版类与现有的元素相差不多，只不过比现有的元素多一些特别的语义罢了。版类与产生该版类的原始元素的使用场所是一样的。版类可以建立在所有的元素类型上，比如：类、结点、组件、关系。UML 语言中已经预定义了一些版类，这些预定义的版类可以直接使用，从而免去了再定义新版类的麻烦，使得UML 语言用起来比较简单。模型元素有很多性质，性质用名字和值一对信息表示。性质也称为加标签值。UML 语言中已经预定义了一定数量的性质，用户还可以为元素定义一些附加信息，即定义性质。任何一种类型的信息都可以定义为元素的性质，比如，具体的方法信息、建模进展状况的管理信息、其他工具使用的信息、用户需要给元素附加的其他各类的信息。约束是对元素的限制。通过约束限定元素的用法或元素的语义。如果在几个图中都要使用某个约束，可以在工具中声明该约束，当然也可以在图中边定义边使用。

4. 答：UML通用划分包含下面两种情况：

（1）. 类/对象二分法（class/object dichotomy），类是一个抽象；对象是这种抽象的一个具体形式。UML的每一个构造块几乎都存在像类/对象这样的二分法。例如：用例和用例实例（场景），构件和构件实例，节点和节点实例等。

（2）. 接口/实现二分法（interface/realization dichotomy），接口声明了一个契约，而实现则表示了对该契约的具体实施，它负责如实地实现接口的完整语义。几乎每一个UML的构造块都有像接口/实现这样的二分法。例如：用例和实现它们的协作，操作和实现它们的方法。

5. 答：聚合（aggregation）是关联关系的一种特例，他体现的是整体与部分、拥有的关系，即has-a的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享；比如计算机与CPU、公司与员工的关系等；表现在代码层面，聚合关系和关联关系是一致的，只能从语义级别来区分。聚合关系用尾部为空心菱形的实线表示（菱形指向整体），如图所示。



聚合关系

组合（compostion）也是关联关系的一种特例，体现的是一种contain-a关系，比聚合更强，是一种强聚合关系。它同样体现整体与部分的关系，但此时整体与部分是不可分的，整体生命周期的结束也意味着部分生命周期的结束，反之亦然。如大脑和人类。体现在代码层面，组合关系与关联关系是一致的，只能从语义来区分。组合与聚合几乎完全相同，唯一区别就是对于组合，“部分”不同脱离“整体”单独存在，其生命周期应该是一致的。

组合关系用尾部带实心菱形的实线表示（菱形指向整体），如图所示。



组合关系

## 第3章 习题参考答案

1. 答：需求分析也称为软件需求分析、系统需求分析或需求分析工程等，是开发人员经过深入细致的调研和分析，准确理解用户和项目的功能、性能、可靠性等具体要求，将用户非形式的需求表述转化为完整的需求定义，从而确定系统必须做什么的过程。需求分析要对软件将要解决的问题进行详细的分析，弄清楚问题的要求，包括需要输入什么数据，要得到什么结果，最后应输出什么。在软件工程中，需求分析就是确定要计算机“做什么”，要达到什么样的效果。

2. 答：需求分析阶段，我们要明确系统的职责，范围和边界；确定软件的功能和性能；构建需求模型（用例模型）。

3. 答：用例建模的主要目标是确定系统既定功能及系统环境的模型，它可以作为客户和开发人员之间的契约。用例是贯穿整个系统开发的一条主线。同一个用例模型即为需求工作流程的结果，可当作分析设计工作流程以及测试工作流程的输入使用。

4. 答：（1）确定系统边界；（2）确定参与者；（3）找出所有的用例；（4）确定每个用例的级别；（5）撰写用例的文字描述。

5. 答：识别参与者的问题如下：1. 谁会来使用这个系统？2. 谁会来安装这个系统？3. 谁会来启动这个系统？4. 谁会来维护这个系统？5. 谁会来关闭这个系统？6. 那些系统会来使用这个系统？7. 谁会从这个系统获取信息？8. 谁会给这个系统提供信息？

识别用例的问题： 1. 参与者想要从这个系统获得什么样的功能？2. 这个系统存储信息吗？哪些参与者将建立、读取、更新和删除这些信息？3. 当系统内部状态发生变化时，这个系统需要通知参与者吗？4. 是否有什么外部事件是这个系统需要知道的？当这个外部事件发生时，哪些参与者会通知这个系统？5. 这个系统需要定期执行什么操作吗？6. 当发生了某些重要的外部事件时，这个系统需要自动执行某些操作吗？7. 这个用例的名称够明确吗？是否能够从这个用例的名称，直接判断出它的结果？8. 这个用例会有多样的结果吗？还是这些结果，是在不同的时间点产生的？

6. 答：通常情况下，用例描述包含以下内容：

* 简要说明：对用例作用和目的的简要描述。
* 事件流：事件流包括基本流和备选流。基本流描述的是用例的基本流程，是指用例“正常”运行时的场景。
* 用例场景：同一个用例在实际执行的时候会有很多不同的情况发生，称之为用例场景，也可以说用例场景就是用例的实例。
* 特殊需求：特殊需求指的是一个用例的非功能性需求和设计约束。特殊需求通常是非功能性需求，包括可靠性、性能、可用性和可扩展性等。例如法律或法规方面的需求、应用程序标准和所构建系统的质量属性等。
* 前置条件：执行用例之前系统必须所处的状态。例如，前置条件是要求用户有访问的权限或是要求某个用例必须已经执行完。
* 后置条件：用例执行完毕后系统可能处于的一组状态。例如，要求在某个用例执行完后，必须执行另一个用例。

7. 答：用户、时间和天气。

8. 答：使用错误，超类参与者应该具有子类参与者的公共属性和公共行为。

9. 答：使用错误，扩展关系应用于构建灵活的系统或者系统异常情况建模。

## 第4章 习题参考答案

1. 答：软件系统的静态分析模型描述的是系统所操纵的数据块之间持有的结构上的关系。它们描述数据如何分配到对象之中，这些对象如何分类，以及它们之间可以具有什么关系。类图和对象图是两种最重要的静态模型。UML中的类图和对象图显示了系统的静态结构，其中的类、对象和关联是图形元素的基础。由于类图表达的是系统的静态结构，所以在系统的整个生命周期中，这种描述都是有效的。

2. 答：类图（Class diagram）是用类和它们之间的关系描述系统的一种图形，是从静态角度表示系统的。类图中的关键元素是类元及它们之间的关系。类元是描述事物的建模元素，类和接口都是类元。类之间的关系包括依赖(Dependency)关系、泛化(Generalization)关系、关联(Association)关系以及实现(Realization)关系等。

3. 关联关系是一种结构关系，指出了一个事物的对象与另一个事物的对象之间在语义上的连接。例如，一个学生选修一门特定的课程。对于构建复杂系统的模型来说，能够从需求分析中抽象出类和类与类之间的关联关系是很重要的。

4. 答：关联的两端都可以指定一个重数，该重数表示该端点可以有多少个对象与另一个端点的一个对象关联。关联关系端点的重数可以描述为取值范围、特定值、无限定的范围或一组离散值。例如，“0..1”表示“零到1 个对象”，“ 5..17”表示 “5 到17 个对象”，“2”表示“2 个对象”。

5. 答：参考4.3节内容。

## 第5章 习题参考答案

1. 答：对象图提供了系统的一个“快照”，显示在给定时间实际存在的对象以及它们之间的链接。可以为一个系统绘制多个不同的对象图，每个都代表系统在一个给定时刻的状态。对象图展示系统在给定时间持有的数据，这些数据可以表示为各个对象、在这些对象中存储的属性值或者这些对象之间的链接。

2. 答：对象图中包含对象(Object)和链(Link)。其中对象是类的特定实例，链是类之间关系的实例，表示对象之间的特定关系。

3. 答：类图与对象图的区别如表所示。

表 类图与对象图的区别

|  |  |
| --- | --- |
| **类 图** | **对 象 图** |
| 在类中包含三个部分，  分别是类名、类的属性和类的操作 | 对象包含两个部分：对象的  名称和对象的属性 |
| 类的名称栏只包含类名 | 对象的名称栏包含“对象名：类名” |
| 类的属性栏定义了所有属性的特征 | 对象的属性栏定义了属性的当前值 |
| 类中列出了操作 | 对象图中不包含操作内容，因为对属于同一个类的对象，其操作是相同的 |
| 类中使用了关联连接，关联中使用名称、角色以及约束等特征定义 | 对象使用链进行连接，链中包含名称、角色 |
| 类是一类对象的抽象，类不存在多重性 | 对象可以具有多重性 |

4. 答：每个链接都可以看作一个属性：标签或角色，表示属性的名称。因此可以说，aCustomer的属性address把它链接到Address对象上，属性name把它链接到表示其名称的String对象上。箭头表示可导航性，即知道另一个对象在哪里。因为Customer端没有箭头，这表示String不知道它与aCustomer的关联。可导航的链接在面向对象的程序中常常称为指针(指针是对象在内存中的地址，以便在需要时能找到它)。

5. 答：当创建对象的程序不再使用该对象了，该怎么办？这似乎是一个小问题，但程序中的对象都不是免费的：每个新对象都要占用计算机内存的一个小区域，在程序运行时，可能会创建越来越多的对象，这样，运行其他程序的可用内存就会减少。如果在对象使用完后不重新声明，计算机就可能用尽内存(程序使用的内存通常在该程序结束后，返回给计算机，但有可能同时运行好几个程序，其中一些可能运行几天、几星期或几年)。

最好不要让程序创建越来越多的对象，且在它们的生命周期结束后，要采取措施清理它们。传统上，程序员必须确定何时去除与对象的最后连接，以便显式地删除或释放对象的内存(结构化的语言没有对象，但有记录、结构和数组，它们也需要释放)。跟踪对象的生命周期是很复杂的，程序员很容易忘记一些已没有用的对象，使问题更严重，这种错误称为内存泄漏。

像Java这样的语言规定，程序会自动重新声明对象，程序员不需要做任何事。其理念是每个程序都有一个助手，称为垃圾收集器。它四处巡视，查找未连接的对象，并清理它们。听起来很神奇吧？实际上并非如此。现在，每个程序都有一个运行时系统(run-time system)，这个软件总是在我们编写的代码后面执行，它执行内务操作，例如垃圾收集。

6. 答：参考5.3.4节内容。

## 第6章 习题参考答案

1. 答：序列图描述了对象之间传递消息的时间顺序，用来表示用例中的行为顺序。序列图的主要用途之一是从一定程度上更加详细地描述用例表达的需求，并将其转化为进一步、更加正式层次的精细表达。

2. 答：序列图(Sequence Diagram)是由对象(Object)、生命线(Lifeline)、激活(Activation)和消息(Messages)等构成的。序列图描述了对象以及对象之间传递的消息，强调对象之间的交互是按照时间的先后顺序发生的，这些特定顺序发生的交互序列从开始到结束需要一定的时间。

3. 答：对象生命线上的窄矩形条被称为激活(Activation)，激活表示该对象正在执行某个操作。激活条的长短表示执行操作的时间。一个被激活的对象要么执行自己的代码，要么等待另一个对象的返回结果。

4. 答：

* 决定交互建模的层面，是否描述了操作、用例、组件之间的消息传递、子系统或系统的交互？
* 确认交互中涉及的主要元素。如果交互发生在用例层面，那么协作对象可能已经通过使用CRC卡以及部分分配职责得以确定。当然，CRC卡可以用于不同层面的粒度，以挖掘任何生命线分组的行为。
* 考虑可能需要的替代场景。通常在研究替代场景的时候，CRC卡会很有用。
* 确认已经或即将建模为序列图的所有现有交互，以便它们可以作为交互使用被包含其中。
* 绘制图的大纲结构。
* 使用合适的名称创建框图。
* 添加合适的生命线，从最先在交互中涉及的生命线开始，然后从左向右放置其他的生命线。这可以改善序列图的页面布局。如果执行者生命线已经被建模，那么应该紧跟着边界生命线(如果已经被建模的话)被放置。
* 添加详细交互。
* 从框图的最上面开始添加首条消息。从上至下布局接下来的消息，在消息标签中显示合适的细节层面。
* 使用带有合适交互操作符的组合片段来进行描述，例如循环、分支和选择路径。交互操作符的完整列表如表9-1所示。需要的话，添加交互约束。
* 确认在其他交互中正在或即将使用的所有交互片段，将它们布置在单独的序列图中。对于这些交互片段准备序列图，以便尽可能进行重用。在绘制的图中布置对应的交互使用。
* 需要的话，注释图，例如包括前置或后置条件以提高可读性。
* 在图中按照需要添加状态变量。
* 检查与链接的序列图的一致性，需要的话进行修改。如果交互是在用例层面进行的，考虑由扩展或包含依赖关系链接的其他用例是非常有用的。
* 检查与其他UML图或模型的一致性，特别是与相关类图的一致性(如果此时已经准备的话，检查与状态机图的一致性)。

5. 答：略。

6. 答：略。

## 第7章 习题参考答案

1. 答：协作图是对在一次交互过程中有意义对象和对象间的链建模，显示了对象之间如何进行交互以执行特定用例或用例中特定部分的行为。在协作图中，类元角色描述了一个对象，关联角色描述了协作关系中的链，并通过几何排列表现交互作用中的各个角色。

2. 答：协作图（Collaboration Diagram）是由对象（Object）、消息（Messages）和链（Link）等构成的。协作图通过各个对象之间的组织交互关系以及对象彼此之间的链接，表达对象之间的交互。

3. 答：协作图与序列图有很多相似的地方。对于直接交互来说，协作图以不同的格式表达了与序列图相同的信息，它们可能在各种细节层面以及系统开发过程的不同阶段绘制。对于这两种类型的交互模型，最显著的区别在于：协作图明确显示了参与协作的生命线之间的链接。不同于序列图，协作图没有明确的时间维度，生命线只是使用方框表示。

4. 答：略。

5. 答：二者描述了同一用例场景。

## 第8章 习题参考答案

1. 答：状态机是一种记录下给定时刻状态的设备，它可以根据各种不同的输入对每个给定的变化而改变其状态或引发一个动作，如计算机、各种客户端软件、Web上的各种交互页面都是状态机。

在UML中状态机由对象的各个状态和连接这些状态的转换组成，是展示状态与状态转换的图。在面向对象的软件系统中，一个对象无论多么简单或者多么复杂，都必然会经历一个从开始创建到最终消亡的完整过程，这个过程通常被称为对象的生命周期。一般说来，对象在其生命周期内是不可能完全孤立的，它必然会接受消息来改变自身或者发送消息来影响其他对象。而状态机就是用于说明对象在其生命周期中响应事件所经历的状态序列以及对这些事件的响应。在状态机的语境中，一个事件就是一次激发的产生，每个激发都可以触发一个状态转换。

通常一个状态机依附于一个类，并且描述该类的实例(即对象)对接收到的事件的响应。除此之外，状态机还可以依附于用例、操作等，用于描述它们的动态执行过程。在依附于某个类的状态机中，总是将对象孤立地从系统中抽象出来进行观察，而将来自外部的影响都抽象为事件。

在UML中状态机常用于对模型元素的动态行为进行建模，更具体地说就是对系统行为中受事件驱动的方面进行建模。不过状态机总是一个对象、协作或用例的局部视图。由于它考虑问题时将实体与外部世界相互分离，所以适合对局部、细节进行建模。

2. 答：状态图本质上就是一个状态机，或者是状态机的特殊情况，它本质上是一个状态机中的元素的投影，这也就意味着状态图包括状态机的所有特征。状态图主要用来描述一个特定对象的所有可能状态以及由于各种事件的发生而引起状态之间的转换。通过状态图可以知道一个对象、子系统、系统的各种状态及其收到的消息对其状态的影响。通常创建一个UML状态图是为了以下的研究目的：研究类、角色、子系统或构件的复杂行为。

3. 答：状态图主要用于描述一个对象在其生存期间的动态行为，表现为一个对象所经历的状态序列，引起状态转移的事件（Event），以及因状态转移而伴随的动作（Action）。一般可以用状态机对一个对象的生命周期建模，状态图用于显示状态机（State Machine Diagram），重点在与描述状态图的控制流。

4. 答：简单状态不能分解，没有子结构。在简单状态之外，还有一种可以包含嵌套子状态的状态，又称为组成状态。在复杂的应用中，当状态图处于某种特定的状态时，状态图描述的该对象行为仍可以用另一个状态图描述，用于描述对象行为的状态图又称为子状态。当对象或交互的状态行为比较复杂时，在不同的细节层面表示它们并且反映在应用程序中出现的任何层级结构，可能是合适的。

5. 答：如果一个组成状态的子状态对应的对象在其生命周期内的任何时刻都只能处于一个子状态，也就是说状态图中多个子状态是互斥的，不能同时存在，这种组成状态称为顺序组成状态。在顺序组成状态中最多只能有一个初态和一个终态。

当状态图通过转换从某种状态转入组合状态时，该转换的目的可能是组成状态本身，也可能是这个组成状态的子状态。如果是组成状态本身，先执行组合状态的入口动作，然后子状态进入初始状态并以此为起点开始运行；如果转换的目的是组合状态的某一子状态，那么先执行组合状态的入口动作，然后以目标子状态为起点开始运行。有时组成状态有两个或多个并发的子状态，此时称该组成状态为并发组成状态。并发组成状态能说明很多事情发生在同一时刻，为了分离不同的活动，组成状态被分解成区域，每个区域都包含一个不同的状态图，各个状态图在同一时刻分别运行。

如果并发组成状态中有一个子状态比其他并发子状态先到达它的终态，那么先到的子状态的控制流将在它的终态等待，直到所有的子状态都到达终点。此时，所有子状态的控制流汇合成一个控制流，转换到下一个状态。

6. 答：略。

## 第9章 习题参考答案

1. 答：活动图是一种用于描述系统行为的模型视图，它可用来描述动作和动作导致对象状态改变的结果，而不用考虑引发状态改变的事件。通常，活动图记录单个操作或方法的逻辑、单个用例或商业过程的逻辑流程。活动图允许读者了解系统的执行，以及如何根据不同的条件和触发改变执行方向。

2. 答：动作通常用于对工作流执行过程中的步骤进行建模。在一张活动图中，动作状态允许在多处出现。不过动作和状态图中的状态不同，它不能有入口动作和出口动作，也不能有内部转移。在UML中，动作状态使用平滑的圆角矩形表示，动作状态表示的动作写在矩形内部，如图所示。



3. 答：略。

4. 答：略。

5. 答：略。

6. 答：分支在活动图中很常见，它是转换的一部分，它将转换路径分成多个部分，每一部分都有单独的监护条件和不同的结果。当动作流遇到分支时，会根据监护条件(布尔值)的真假来判定动作的流向。分支的每个路径的监护条件应该是互斥的，这样可以保证只有一条路径的转换被激发。

7. 答：活动图中交互的简单元素是活动和对象，控制流(Control Flow)就是对活动和对象之间的关系的描述。控制流表示动作与其参与者和后继动作之间、动作与输入和输出对象之间的关系。而对象流就是一种特殊的控制流。

对象流(Object Flow)是将对象流状态作为输入或输出的控制流。在活动图中，对象流描述了动作状态或者活动状态与对象之间的关系，表示了动作使用对象以及动作对对象的影响。

8. 答：对象流表示了对象与对象、操作或产生它（使用它）的转换间的关系。在活动图中，用带箭头的虚线而非实线来表示对象流。如果虚线箭头从活动指向对象流状态，则表示输出。输出表示了动作对对象施加了影响，影响包括创建、修改、撤销等。如果虚线箭头从对象流状态指向活动，则表示输入。输入表示动作使用了对象流所指向的对象流状态。如果活动有多个输出值或后继控制流，那么箭头背向分叉符号。反之，如果有多个输入箭头，则指向结合符号。

## 第10章 习题参考答案

1. 答：UML提供了包图、构件图、部署图等三种模型图，用于对系统的设计进行归档。包图用于对系统的层次结构进行描述。构件图用于归档系统中不同元素之间的依赖关系。它们可以与部署图结合，显示软件构件和系统的物理架构是如何关联的。

2. 答：设计阶段的主要任务包括：

* 系统体系结构设计
* 数据结构设计
* 用户界面设计
* 算法设计

3. 答：构件图（Component Diagram）描述软件构件及构件之间的关系，显示代码的结构。构件是逻辑架构中定义的概念和功能(类、对象、它们的关系、协作)在物理架构中的实现。构件图用于显示组成系统的逻辑或物理软件构件。

4. 答：构件是独立的，是在一个系统或子系统中的封装单位，提供一个或多个接口，是系统高层的可重用的部件。构件作为系统中的一个物理实现单元，包括软件代码(包括源代码、二进制代码和可执行文件等)或者相应组成部分，例如脚本或命令行文件等，还包括带有身份标识并有物理实体的文件，如运行时的对象、文档、数据库等。

采用标准化构件的原因众所周知，并且显而易见，实际上，难以想象哪个行业没有广泛使用它们。例如，房屋通常是由砖块、屋顶木材、瓷砖、门、窗框、电器元件、排水管道、地板等等设计和组建的，这些都是从建筑材料中挑选出来的。建筑师可能使用这些构件去设计(或者由建筑工人去建造)整体形象、平面图和房间数量方面与众不同的房屋。区别在于标准构件集成的方式。

对于被当作构件的所有事物来说(例如窗户)，必须按照一定的方式被规范，以允许建筑师、建筑工人等等作为单个简单的事情来完成，即便实际并非如此。另外，专业设计师处理窗户设计的问题(采用什么材料和结构等等，从而制作好的窗户)，专业制造师将之组建为设计者的规范。标准窗户的构造细节可能会因为设计的改善、可使用的材料或者新的建筑方法而改变。然而只要诸如高度、宽度和整体外观之类的关键特征没有改变——实际上，窗户的接口——在将来需要的时候，新的窗户就能够替换相同类型的旧窗户。

同样，软件构件对请求其服务的其他构件来说，会隐藏实现细节。因此，不同的子系统在操作上会有效地分离。这大大降低了互相交互造成的问题，即便它们是在不同的时间，使用不同的语言开发的，或者在不同的硬件平台上执行的；也允许一个构件可以被另一构件替换，只要二者具有相同的接口即可。按照这种方式规定的子系统被认为是互相解耦合的。

这一方法可以被扩展到任意复杂度级别。假定满足如下指标，软件系统的任何部分——或者模型的任何部分，在一些情况下都将被认为是可重用的：构件满足明确且通用的需求(换言之，发布了连贯一致的服务或服务集)。构件具有一个或多个简单的、定义完好的外部接口。

5. 答：节点是拥有某些计算资源的物理对象(设备)。 这些资源包括：带处理器的计算机，一些设备如打印机、读卡机、通信设备等等。在查找或确定实现系统所需的硬件资源时标识这些节点，主要描述节点两方面的内容：能力(如基本内存计算能力二级存储器)和位置(在所有必须的地理位置上均可得到)。

节点的确定可以通过查看对实现系统有用的硬件资源来完成，需要从能力(如计算能力、内存大小等)和物理位置(要求在所有需要使用该系统的地理位置都可以访问该系统)两方面来考虑。

节点用带有节点名称的立方体表示，节点的名称是一个字符串，有两种：简单名和路径名；位于节点图标内部。实际应用中，节点名称通常是从现实的词汇表中抽取出来的短名词或名词短语。通常，UML图中的节点只显示名称，也可以用标记值或表示节点细节的附加栏加以修饰。

在Rational Rose中可以表示的节点类型包括两种，分别是处理器(Processor)节点和设备(Device)节点。

6. 答：部署图描述处理器、设备、软件构件在运行时的架构。它是系统拓朴的最终的物理描述，即描述硬件单元和运行在硬件单元上的软件的结构。在这样的架构中，在拓朴图中寻找一个指定节点是可能的，从而了解哪一个构件正在该节点上运行，哪些逻辑元素(类、对象、协作等等)是在本构件中实现的，并且最终可以跟踪到这些元素在系统的初始需求说明(在用例建模中完成的)中的位置。

## 第11章 习题参考答案

1. 答：软件，尤其是许多人一起开发的大型软件，应使用某种方法来开发。甚至由一个人开发的小型软件也应通过某种方法进行改进。方法是对完成一项工作所需步骤的逐步描述。因为没有两个项目是完全一样的，任何方法都针对特定的项目。方法论是指导参与者或经理选择适用于特定任务或项目的特定方法的一组通用原则。

方法学是做事的系统方法，它是一个可接受的过程，从软件开发的早期阶段(有一个想法或一个新的商业机会)到已安装系统的维护，都可以遵循这个方法学。除了过程之外，方法学还应指定在此过程中要生产什么产品(以及该产品采用什么形式)。方法学也包括用于资源管理、规划、调度和其他管理任务的建议或技术。优秀的、适用范围广的软件开发方法学是成熟软件业的基础。

2. 答：瀑布模型核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作，即采用结构化的分析与设计方法将逻辑实现与物理实现分开。瀑布模型将软件生命周期划分为项目计划、需求分析、软件设计、程序编写、软件测试和运行维护等六个基本阶段，并且规定了它们自上而下、相互衔接的固定次序，开发进程从一个阶段“流动”到下一个阶段，如同瀑布流水，逐级下落，这也是瀑布模型名称的由来。从本质来讲，它是一个软件开发架构，开发过程是通过一系列阶段顺序展开的，从系统需求分析开始直到产品发布和维护，每个阶段都会产生循环反馈，因此，如果有信息未被覆盖或者发现了问题，那么最好 “返回”上一个阶段并进行适当的修改。

瀑布模型学有很多优点，可使得开发人员采用规范的方法；严格地规定了每个阶段必须提交的文档；要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证。但是，对于瀑布模型而言也有明显的不足。比如，它各个阶段间具有顺序性和依赖性，即下一个阶段的开始对上一个阶段的成果依赖性非常大，所以如果模型某一阶段出现了延迟完成的情况，将会影像以后各阶段的完成。另外，瀑布模型是由文档驱动的，由于瀑布模型几乎完全依赖于书面的规格说明，很可能导致最终开发出的软件产品不能真正满足用户的需要。瀑布方法学是一个很好的想法，但不切实际。即使我们能确定开发需要的时间，在没有考虑问题的细节之前，是不可能预知在开发过程中会遇到什么困难的(糟糕的设计决策、有害的错误、技术不适当或地震)。所以，任何阶段都可能比预期的时间长。另外，工作也可能会扩展，以充分利用可用的时间，这样，某个问题之前的各个阶段很可能用尽了所有的可用时间。最终结果是整个项目都得延迟交付。实际上，这就是大多数项目的具体情况。

瀑布方法学也可能因其他原因而失败，例如分析失效—— 分析员不愿签署其文档，因为他们不能确保已经很好地理解了系统实体，能让设计人员完成其工作，并为它们编写文档。而且，这类问题并不限于分析员：设计人员也可能担心他们的设计不适当；规范人员可能担心他们的规范过于模糊，不能用于编程等，这些会导致更多的延迟。实际上，根本不可能非常完美地完成每个阶段。在整个开发过程中，小组成员会发现他们已完成的工作中有问题。只要出现这种情况，我们就面临着一个两难的选择：要么返回到前面的说明文档中，更正错误，但这意味着逆着瀑布向上走(不推荐这么做)；要么记下问题，在项目的最后修订说明文档(这是很少发生的，所以最后的说明文档不匹配最终系统)。

3. 答：统一过程RUP是一套软件工程方法，是Rational 软件公司的软件工程过程框架，主要由Ivar Jacobson的The Objectors Approach和The Rational Approach发展而来。它定义了进行软件开发的工作步骤，亦即定义了软件开发过程中的，什么时候做，做什么，怎么做，谁来做的问题，以保证软件项目有序地、可控地、高质量地完成。 RUP凭借Booch、Ivar Jacobson以及Rumbagh在业界的领导地位，与统一建模语言（Unified Model Language，简称UML）的良好集成、多种CASE工具的支持、不断的升级与维护，迅速得到业界广泛的认同，越来越多的组织以它作为软件开发模型框架。

RUP是面向对象开发技术发展的产物，这种方法旨在将面向对象技术应用于软件开发的所有过程，包括需求分析、系统分析、系统设计、系统实现和系统升级维护等所有过程，使软件系统开发的所有过程全面结合，最大限度适应用户不断变化的需求，有效地降低风险，更好地适应需求变化，因此软件研发人员经常采用RUP来指导项目开发的全过程。

4. 答：略。

5. 答：RUP中的软件生命周期在时间上被分解为四个顺序的阶段，分别是：初始阶段(Inception)、细化阶段(Elaboration)、构造阶段(Construction)和交付阶段(Transition)。

6. 答：(1) 商业建模（Business Modeling）

该工作流的主要目的是对系统的商业环境和范围进行建模，确保所有参与人员对开发系统有共同的认识，并在商业[用例模型](http://baike.baidu.com/view/977665.htm)和商业对象模型中定义组织的过程，角色和责任。

(2) 需求分析（Requirements）

需求（Requirement）[工作流](http://baike.baidu.com/view/60285.htm)的目标是描述系统应该做什么，并使开发人员和用户就这一描述达成共识。为了达到该目标，要对需要的功能和约束进行提取、组织、文档化；最重要的是定义系统功能及用户界面，明确可以需要的系统的功能。

(3) 分析与设计（Analysis and Design）

分析和设计（Analysis & Design）[工作流](http://baike.baidu.com/view/60285.htm)将[需求](http://baike.baidu.com/view/195818.htm)转化成未来系统的设计。为系统开发一个健壮的结构并调整设计使其与实现环境相匹配，优化其性能。分析设计的结果是一个设计模型和一个可选的分析模型。设计模型是[源代码](http://baike.baidu.com/view/60376.htm)的抽象，由设计类和一些描述组成。设计类被组织成具有良好接口的设计包（Package）和设计子系统（Subsystem），而描述则体现了类的对象如何[协同工作](http://baike.baidu.com/view/8202227.htm)实现[用例](http://baike.baidu.com/view/706238.htm)的功能。设计活动以[体系结构设计](http://baike.baidu.com/view/553707.htm)为中心，体系结构由若干结构视图来表达，结构视图是整个设计的抽象和简化，该视图中省略了一些细节，使重要的特点体现得更加清晰。体系结构不仅仅是良好设计模型的承载媒介，而且在系统的开发中能提高被创建模型的质量。

(4) 实现（Implementation）

实现工作流包含定义代码的组织结构、实现代码、单元测试和系统集成四个方面的内容。实现工作流的目的包括以层次化的子系统形式定义代码的组织结构，以组件的形式(源文件、二进制文件、可执行文件)实现类和对象。将开发出的组件作为单元进行测试，以及集成由单个开发者所开发的组件，使其成为可执行的系统。

(5) 测试（Test）

测试（Test）[工作流](http://baike.baidu.com/view/60285.htm)要验证对象间的交互作用，验证[软件](http://baike.baidu.com/view/37.htm)中所有组件的正确集成，检验所有的[需求](http://baike.baidu.com/view/195818.htm)已被正确的实现，识别并确认缺陷在软件部署之前被提出并处理。RUP提出了迭代的方法，意味着在整个项目中进行测试，从而尽可能早地发现缺陷，从根本上降低了修改缺陷的成本。测试类似于三维模型，分别从可靠性、功能性和系统性能来进行。

(6 )配置（Deployment）

部署（Deployment）[工作流](http://baike.baidu.com/view/60285.htm)的目的是成功的生成版本并将[软件](http://baike.baidu.com/view/37.htm)分发给最终用户。部署[工作流](http://baike.baidu.com/view/60285.htm)描述了那些与确保[软件](http://baike.baidu.com/view/37.htm)产品对最终用户具有可用性相关的活动，包括：软件打包、生成软件本身以外的产品、安装软件、为用户提供帮助。在有些情况下，还可能包括计划和进行beta测试版、移植现有的[软件](http://baike.baidu.com/view/37.htm)和数据以及正式验收。

7. 答：RUP是过程组件、方法以及技术的框架，可以应用于任何特定的软件项目，由用户自己限定RUP的使用范围。其应用优势在于：

* 用例驱动：架构采用用例驱动，能够更有效地从需求转到后续的分析和设计；
* 增量迭代：采用迭代和增量式的开发模式，便于相关人员从迭代中学习；
* 协同工作：统一过程是一个工程化的过程，所以它能使项目组的每个成员协调一致的工作；并从多方面强化了软件开发组织。最重要的是它提供了项目组可以协同工作的途径；
* 项目间协调：RUP提供了项目组与用户及其他项目相关人员一起工作的途径。
* 易于控制：RUP的重复迭代和用例驱动、体系结构为中心的开发使得开发人员能比较容易地控制整个系统的开发过程，管理其复杂性并维护其完整性。
* 易于管理：体系结构中定义清晰、功能明确的组件为基于组件式的开发和大规模的软件复用提供了有力的支持，也是项目管理中计划与人员安排的依据。
* 工具丰富：RUP辅助以Rational公司提供的丰富的支持RUP的工具集，包括可视化建模工具RationalRose、需求管理工具RequisitePro、版本管理工具ClearCase、文档生成SoDa、测试工具SQA和Performance等；由于RUP采用标准的UML描述系统的模型体系结构，因此可以利用很多第三方厂家提供的产品。

但是RUP只是一个开发过程，并没有涵盖软件过程的全部内容，例如它缺少关于软件运行和支持等方面的内容；此外，对于各种类型的软件项目，RUP并未给出具体的自身裁减及实施策略，降低了在开发组织内大范围实现重用的可能性。RUP适用于规模比较大的软件项目和大型的软件开发组织或团队，提供了在软件开发中涉及到的几乎所有方面的内容。但是，对于中、小规模的软件项目，开发团队的规模不是很大，软件的开发周期也比较短。在这种情况下，完全照搬RUP并不完全适用。

8. 答：参考11.4节内容。